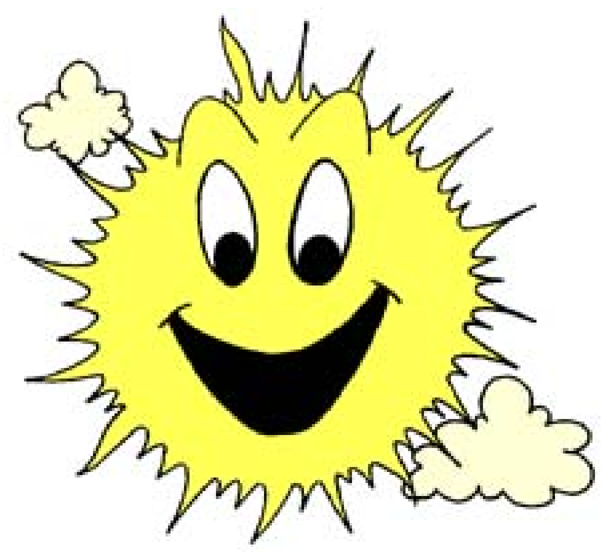
Wiederholung der 7. Vorlesung



- Von Neumann Architektur
  - Ein- und Ausgabeeinheit, Speicher
  - ALU
  - Kontrolleinheit

6.1 von Neumann Architektur

Speicher
(Memory)

Eingangseinheit
(input unit)

Arithmetisch/
Logische Einheit
(ALU)

Ausgangseinheit
(output unit)

Kontrolleinheit
(control unit)

Datenleitungen
Kontrollleitungen

6.1 von Neumann

von Neumann-Rechner besteht aus 5 Funktionen:

1. Eingangseinheit

   oder Eingabewerk dient zur Aufnahme von Daten in den Rechner

2. Ausgabeeinheit

   oder Ausgabewerk dient der Ausgabe der vom Rechner ermittelten Ergebnisse

3. Speicher
4. Rechenwerk/ALU
5. Kontrolleinheit

6.1.3 Speicher

- Im Speicher werden Daten abgelegt. Der Speicher hat eine Informationsstruktur, die auf „**Worten**" basiert. Ein Wort ist eine Binärzahl fester Länge.
- Die Anzahl der Bits eines Speicherworts wird als **Speicherwortbreite** bezeichnet. Typische Wortbreiten sind 8, 16, 32 oder 64 Bit.
- Im Speicher wird keinerlei Unterscheidung zwischen verschiedenen Datentypen vorgenommen. So werden z.B. **Instruktionen** und **Programmdaten** in der gleichen Informationsstruktur (Wort) des Speichers abgelegt.

6.1.4 Rechenwerk / ALU

- Diese Einheit führt alle Berechnungen des Programms durch. Dazu beinhaltet diese Einheit arithmetische und logische Verarbeitungseinheiten zur Durchführung von arithmetischen und logischen Operationen sowie von Schiebeoperationen. (**Arithmetic Logical Unit**)
- Nach Durchführung einer Operation stehen sowohl das Ergebnis als auch sogenannte Bedingungs-Bits (Flags) zur Verfügung.
- Die Bedingungs-Bits beschreiben das Ergebnis der Operation (z.B. Null, Negativ, Überlauf, Carry).

6.1.4 Steuerung der ALU

Op1[23..0]
Op2[23..0]
AluFkt[1..0]
Konstante
Logik-funktionen
Schieben und Durchreichen
Arithmetik-funktionen
O
C
AluFkt[3..0]
Alu0[23..0]
Alu1[23..0]
Alu2[23..0]
Alu3[23..0]
AluFkt[3..2]
0
1
2
3
AluCC[3..0]
Vergleichs-einheit
N
Z
Ergebnis[23..0]

Die 4 Funktionsblöcke arbeiten parallel, durch einen nachgeschalteten Multiplexer wird das ALU-Ergebnis von der durch AluFkt[3..2] selektierten Funktionseinheit ausgewählt.

6.1.5 Kontrolleinheit / Leit- / Steuerwerk

Eines der wichtigsten allgemeinen Entwurfsprinzipien ist die **Trennung** von **Kontrolle** und der **Verarbeitung von Daten**, dementsprechend beim Hardware-Entwurf die Trennung von Steuerwerk und Operationswerk:

Die Kontrolleinheit (Steuerwerk, Leitwerk) interpretiert die im Speicher abgelegten Worte. Je nach Interpretation sind dies:

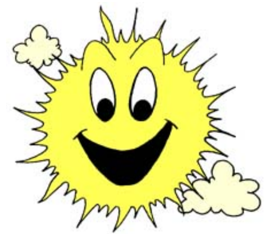
- Instruktionen (Befehle),
- Programmdaten oder
- Adressen

6.1.5 Kontrolleinheit / Leit- / Steuerwerk

- Aufgrund der Instruktionen werden die übrigen Einheiten gesteuert, so dass die Instruktionen korrekt ausgeführt werden. In der Kontrolleinheit wird der Zustand des Rechners gehalten, der die Speicherposition der nächsten Instruktion bestimmt.
- Ein Register in der Kontrolleinheit zeigt auf die nächste auszuführende Instruktion → **Programmzähler** (program counter, **PC**). Nach Ausführung einer Instruktion wird der Programmzähler erhöht und zeigt direkt auf das der aktuellen Instruktion nachfolgende Speicherwort.
- Sprungbefehle verändern den Programmzähler und unterbrechen damit die sequentielle Instruktionssequenz.
- Adressberechnungen werden mit einer endlichen Anzahl von Bits ausgeführt. Diese Anzahl wird als **Adresswortbreite** bezeichnet.

6.1.5 Modell eines Befehlsablaufs

Ende der Wiederholung



6.1.6 CPU

In realen Rechnern werden die Arithmetisch/Logische Einheit und die Kontrolleinheit häufig in einer physikalischen Einheit (Chip) zusammengefasst.

Diese Kombination wird als **Zentrale Verarbeitungseinheit** (central processing unit, **CPU**) bezeichnet.

Auch ein Teil des Speichers rutscht in die CPU, es werden Operanden für die ALU und allgemeine Register in der CPU gespeichert und nicht jedes Mal aus dem separaten Speicher geholt.

- Stack-Architektur
- Akkumulator-Architektur
- Allgemeinzweck-Register-Architektur (GPR)

6.1.6 CPU

Beispiel für Codierung von Befehlen:

OP-Code Adressteil

15	0
4 Bit	12 Bit

Es sei der Operationscode eines Befehls 4H und die Adresse der Speicherzelle 80H:

15			12	11											0
0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0

6.1.6 von Neumann Instruktionstypen

Die wichtigsten Instruktionstypen eines „von Neumann“-Rechners sind:

- **Transferbefehle** (load, store, LDA, STA)
- **Arithmetische Befehle** (addieren, subtrahieren), Logische Befehle (AND, OR,XOR), Shift-Befehle.
- **Sprungbefehle** (bedingt, unbedingt, JMP, JZ, JNZ)

6.1.6 Transferbefehle

Transfer: Die Transferbefehle dienen zum Laden von Programmdaten aus dem Speicher in die ALU (load) und zum Speichern von Ergebnissen von der ALU zurück in den Speicher (store).

Nach Ausführung einer Transferinstruktion zeigt der PC auf das nachfolgende Wort im Speicher.

6.1.6 Arithmetische Befehle

Arithmetik, Logik, Schieben: Die arithmetischen und logischen Befehle sowie die Schiebebefehle stellen die Operationen bereit, die für Berechnungen benötigt werden. Nach Ausführung einer Operation stehen das **Ergebnis** und die **Bedingungs-Bits** in der ALU zur Verfügung.

Nach Ausführung der Instruktion zeigt der PC auf das nachfolgende Wort im Speicher.

6.1.6 Sprungbefehle

Sprünge: Bei den Sprungbefehlen wird der PC mit einem neuen Wert geladen, damit wird die sequentielle Instruktionsreihenfolge im Speicher durchbrochen. Sprungbefehle unterteilen sich in unbedingte und in bedingte Sprünge.

- Bei **unbedingten Sprüngen** wird der PC auf jeden Fall mit einem neuen Wert geladen.
- Bei **bedingten Sprüngen** wird der Wert eines zugeordneten Flag-Bits ausgewertet. Ist es gesetzt, wird der PC neu geladen und damit der Sprung ausgeführt. Ansonsten wird der PC wie bei den anderen Befehlen erhöht und zeigt auf das dem Befehl nachfolgende Wort im Speicher.

6.1.6 Prozessorstatus

Die Bedingungs- oder Flagbits eines Prozessors bilden zusammen mit dem PC und ggf. anderen Konfigurationsregistern den so genannten **Prozessorstatus**.

6.1.6 Zwischenbilanz von Neumann-Rechner

Anzahl der Operatoren	Prozentualer Anteil	Beispiel
1	80%	b
2	15%	b+c
3	3%	$a \times b+c$
4	2%	$(a+b) \times (c+d)$
≥5	<<1%	

Anforderungen an einen Prozessor:

- Wenige, einfache Befehle.
- Schnelle Ausführung der Befehle. → Instruktionslängen, um feste Bearbeitungszeiten zu erzielen und damit Pipelining zu ermöglichen. Kein Mikroprogrammsteuerwerk sondern festverdrahtete Logik.
- Viele interne Register, externe Speicherzugriffe minimieren.

→ Hinweis auf RISC Prozessoren und DSPs.

6.1.6 Befehlssatz-Architektur

Um ein Maschinenprogramm für einen bestimmten Rechner zu erstellen, muss man natürlich nicht alle Details der Organisation des Rechnerkerns beherrschen.

Es genügt die Kenntnis der ***Instruktionssatz-Architektur*** (ISA).

Die ISA beinhaltet:

- den Maschinenbefehlssatz (Art und Format der Befehle)
- die adressierbaren Register des Rechnerkerns
- die Darstellungsmöglichkeiten für Daten (Maschinen-Datentypen: Byte, Integer-Zahlen, Fixpunktzahlen etc.)
- die Adressierungsmöglichkeiten des Speichers (Adressierungsmodi)
- die Art der Daten-Ein- und Ausgabe.

6.1.6 Befehlssatz-Architektur (ISA)

Befehlssatzes-Architekturen:

- Stack-Architektur
- Akkumulator-Architektur
- Allgemeinzweck-Register-Architekturen (General Purpose Register Archit.'s – GPR)
  - Register-Register-Maschinen – RISC (auch Load-Store-Architektur genannt)
  - Register-Speicher-Maschinen – CISC
  - Speicher-Speicher-Maschinen

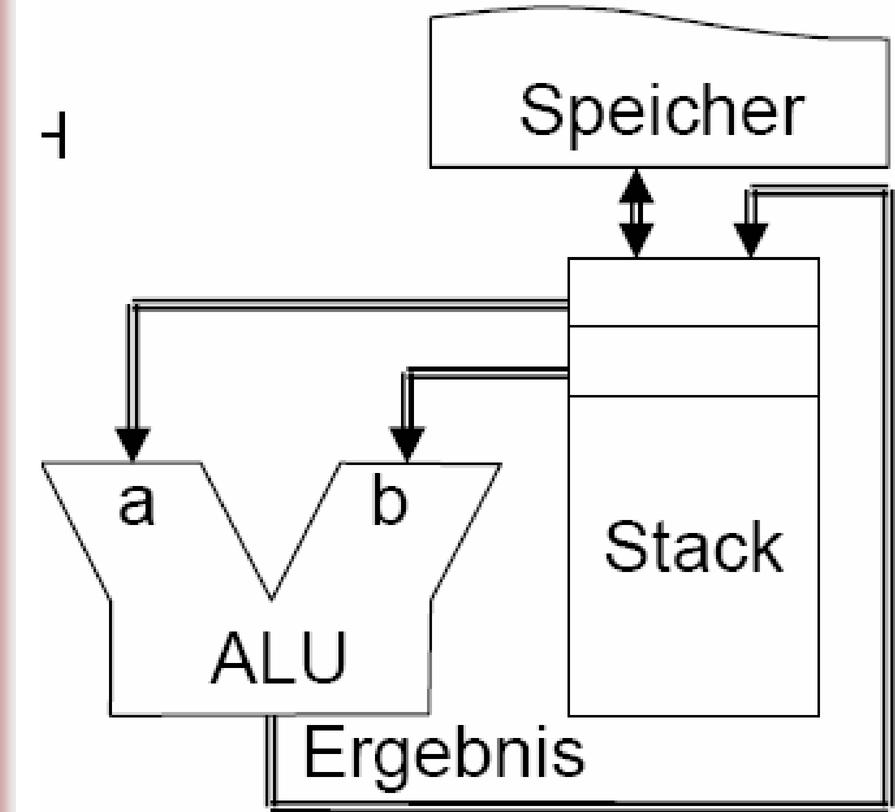
Bei realen Mikroprozessoren wird oft versucht, in eine vorherrschende Befehlssatz-Architektur die Vorteile anderer Befehlssatz–Architekturen zu integrieren (Mischformen).

6.1.6 Stack-Architektur

- CPU hat LIFO-Speicher (Stack), auf den man mit PUSH und POP zugreifen kann
- Es kann meist nur auf die oberste Speicherstelle des Stacks zugegriffen werden
- Sehr einfach, effiziente Codeumsetzung bei geklammerten mathematischen Ausdrücken möglich
- implizite Adressierung von A und B bei ADD

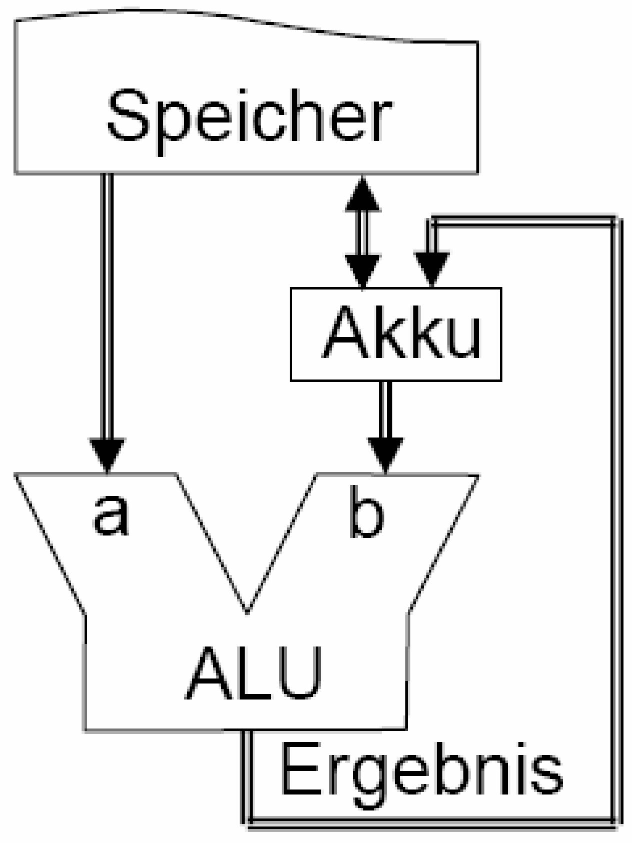
PUSH Op_A ;	Op_A von Speicher -> Stack
PUSH Op_B ;	Op_B von Speicher -> Stack
ADD ;	Addiere obere Werte im Stack
POP Op_C ;	Op_C (= Op_A + Op_B) von Stack -> Sp.

6.1.6 Stack-Architektur



Stacks werden unabhängig von der jeweiligen Architektur bei Unterprogrammaufrufen und Parameterübergaben verwendet. Ein Call Befehl pusht implizit die Rücksprungadresse und ggf. Registerstati, ein Return popt diese Werte wieder.

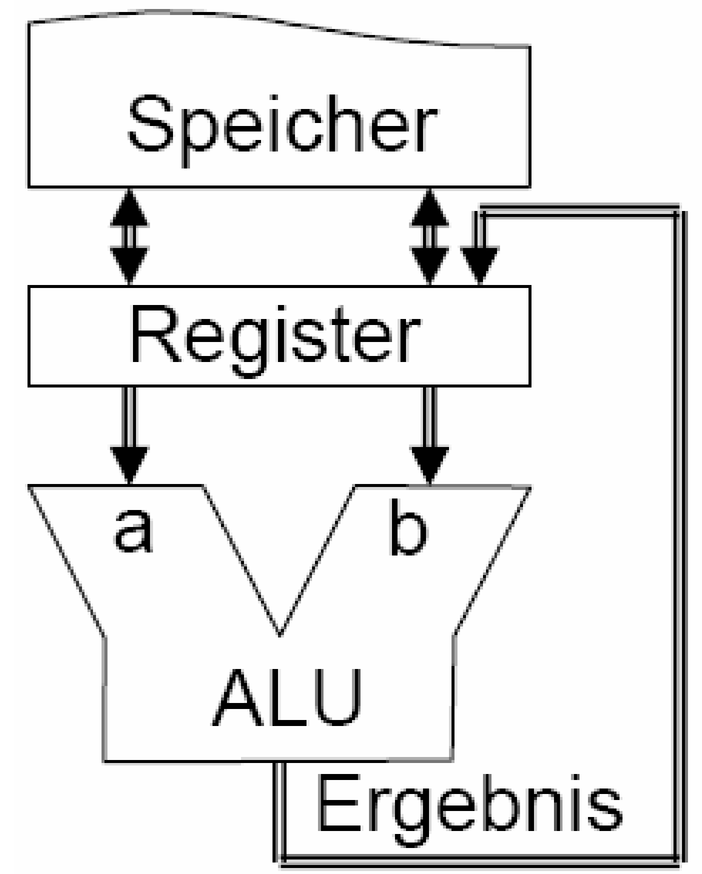
6.1.6 Akkumulator-Architektur



- Ausgezeichnetes Register: Akku
- LOAD und STORE wirken nur auf Akku. Er ist als expliziter Operand an jeder Operation beteiligt. Jede Operation braucht nur eine Adresse
- Sehr kompaktes Befehlsformat

LDA Op_A ;	Op_A von Speicher -> Akku
ADD Op_B ;	Akku = Akku + Op_B
STA Op_C ;	Op_C (= Op_A + Op_B) Akku -> Sp

6.1.6 Register-Register-Architektur

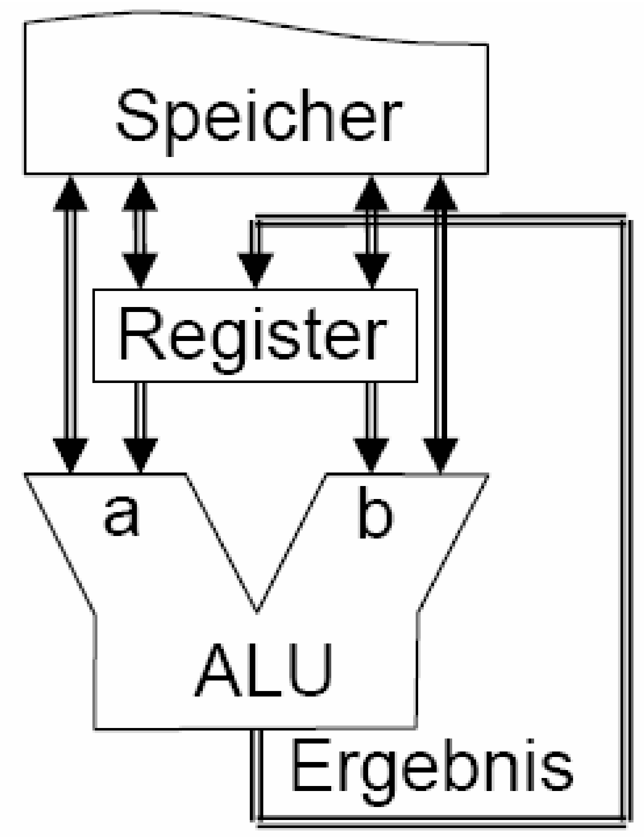


RISC (Load-Store-Architektur)

- alle Operationen greifen nur auf Register zu,
- nur LOAD und STORE greifen auf Speicher zu
- 32 – 512 Register verfügbar
- einfaches Befehlsformat fester Länge
- alle Instruktionen brauchen in etwa gleich lange

LOAD R1, Op_A ; lade Op_A aus Speicher in R1
LOAD R2, Op_B ; lade Op_B aus Speicher in R2
ADD R3, R1, R2 ; addiere R1 und R2, Ergbn. -> R3
STORE Op_C, R3 ;speicher R3 -> Op_C (=Op_A + Op_B)

6.1.6 Register-Speicher-Architektur



CISC (Mischung von Akkumulator- und Load-Store-Architektur)

- Operationen greifen auf Register und/oder Speicher zu
- Befehlsformat variabler Länge
- mächtige Befehle
- stark unterschiedliche Zeiten für Instruktionsausführung

MOV AX, Op_A ; Op_A von Speicher -> Register AX
ADD AX, Op_B ; AX = AX + Op_B
MOV Op_C, AX ; Op_C = AX (= Op_A + Op_B)

6.1.6 Ein von Neumann Beispielrechner

Das nachfolgende Beispiel zeigt die Wirkungsweise eines „von Neumann“-Rechners anhand eines einfachen Beispielrechners.

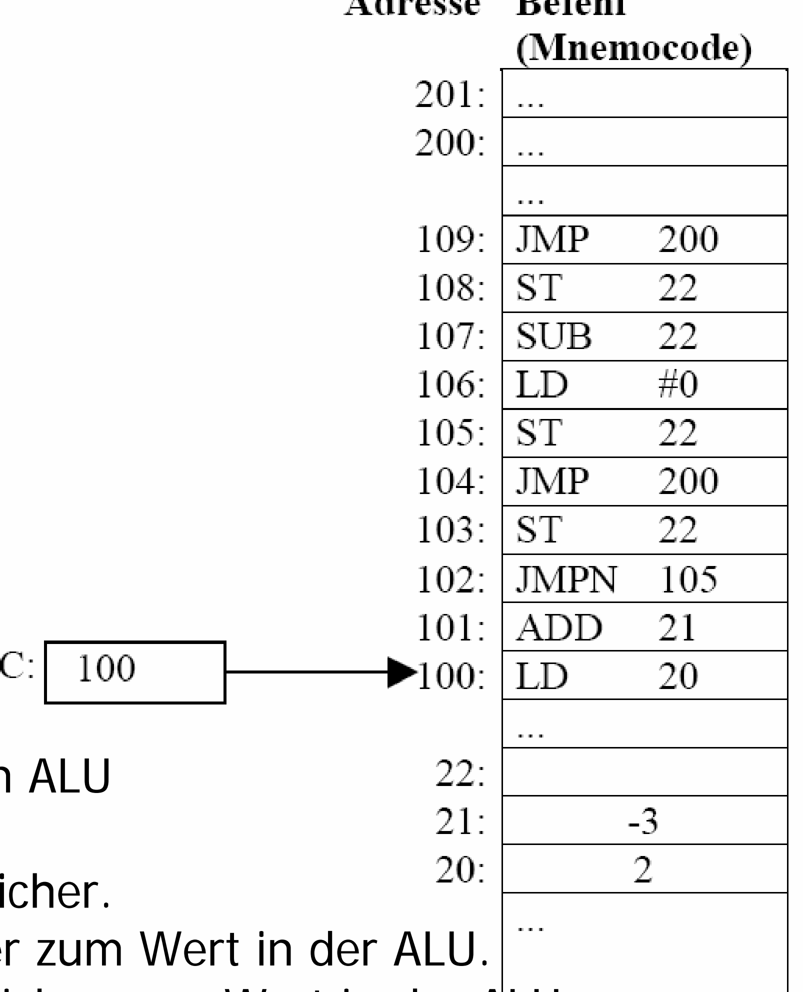
Die nachfolgend verwendeten Befehle des Beispielrechners stellen keineswegs normierte Befehle dar, sie zeigen lediglich eine Möglichkeit der Befehlsimplementierung. Sie wurden so gewählt, daß sie zu Befehlen realer Rechner ähnlich sind und die wichtigsten Instruktionstypen verdeutlichen.

Die Instruktionen werden als **Mnemocodes** dargestellt. Sie stellen eine verständliche Form der Befehle dar und sind direkt abbildbar auf die Bitmuster der Befehle und damit auf die Worte des Speichers.

6.1.6 Ein von Neumann Beispielrechner

Zwei Zahlen sollen addiert und vom Ergebnis der Betrag gebildet werden:

- Die Zahlen stehen im Speicher an den Adressen 20 und 21.
- Das Programmsegment für diese Aufgabe steht ab Adresse 100 im Speicher.
- Nach Ausführung des Programmsegments soll die Ausführung an Adresse 200 fortgesetzt werden.



LD <Adresse> Lade Wert vom Speicher in ALU
LD #<Wert> Lade „Wert" in die ALU
ST <Adresse> Lade Wert von ALU in Speicher.
ADD <Adresse> Addiere Wert vom Speicher zum Wert in der ALU.
SUB <Adresse> Subtrahiere Wert vom Speicher vom Wert in der ALU.
JMP <Adresse> Unbedingter Sprung zu Befehl im Speicher.
JMPN <Adresse> Bedingter Sprung bei Flag „Negativ" zu Befehl im Speicher.

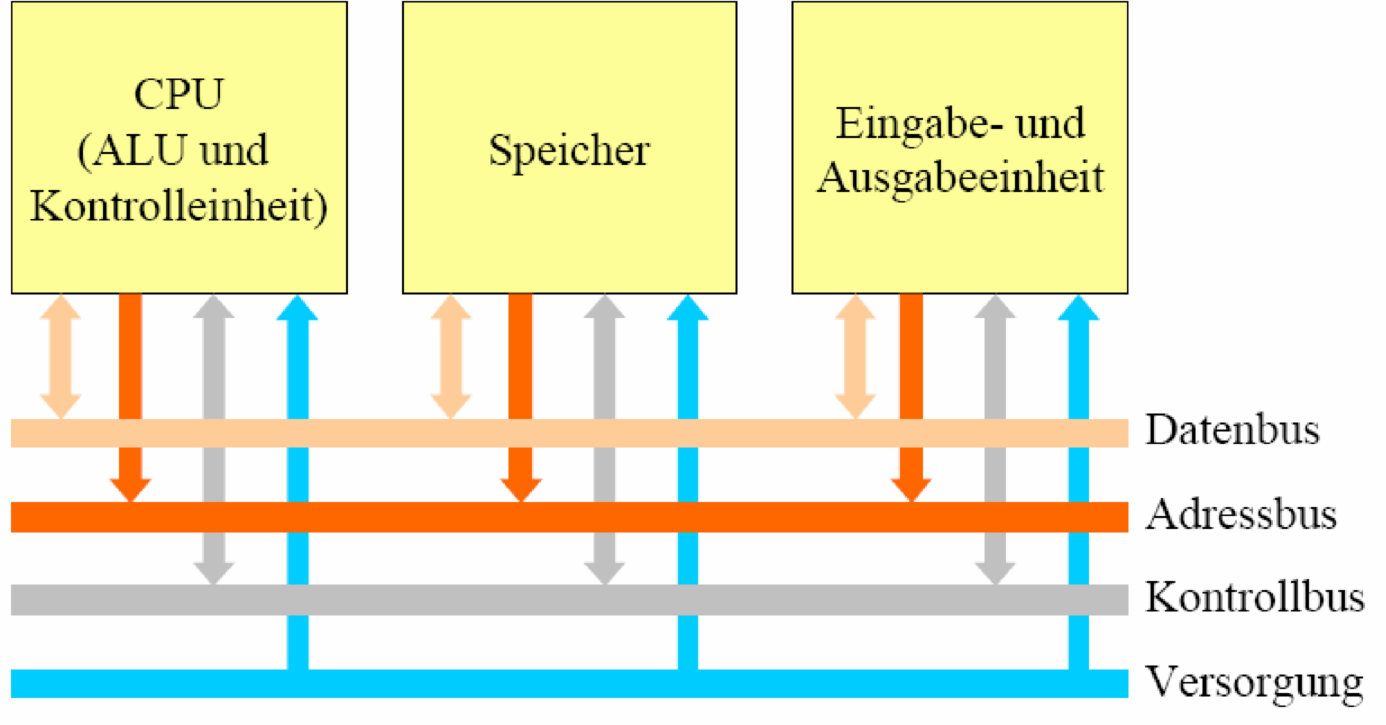
6.1.7 Bussysteme

- Systembus
- Adress- und Datenbus
- Kontrollbus
- Bus zur elektrischen Versorgung
- Buszyklen
  - Synchroner Schreibzyklus
  - Asynchroner Lesezyklus

6.1.7 Bussysteme

Ein **Systembus** ist aus einem **Datenbus**, **Adressbus** und **Kontrollbus** sowie einem Bus zur elektrischen **Versorgung** der Komponenten aufgebaut.

Einige Architekturen beinhalten zusätzlich noch einen I/O-Bus.



6.1.7 Bussysteme

Physikalisch ist jeder Bus aus einer Anzahl von Leitungen aufgebaut.

Der Systembus setzt sich aus mehreren Bussen zusammen, wobei jeder Bus eine individuelle Funktion besitzt.

Zum **Transfer** von Daten zwischen Einheiten werden die Signale der einzelnen Busse in einer spezifizierten Reihenfolge auf den Bussen angelegt.

Für jede Busleitung darf es zu einem Zeitpunkt nur eine Einheit geben, welche die Busleitung treibt und damit einen gültigen Logikpegel auf dem Bus anlegt.

Ansonsten entstehen Kurzschlüsse, die zu Konflikten bei Buszyklen oder schlimmer noch zur Zerstörung von Bustreibern führen können.

6.1.7.1 Adress- und Datenbus

Daten werden von „Quelle“ zur „Senke“ übertragen.

Adress- und **Datenbus** sind homogene Busse, die Signale gleicher Funktion zusammenfassen.

Der **Datenbus** ist ein bidirektionaler Bus:

- Beim Schreibens werden Daten von der CPU (Quelle) zu Speicher oder Ausgabeeinheit (Senke) übertragen.
- Im Falle des Lesens werden Daten in der umgekehrten Richtung von Speicher oder Eingabeeinheit (Quelle) zur CPU (Senke) übertragen.
- Die Anzahl der Datenbusleitungen ergibt sich aus der **Datenbusbreite** eines Rechners, sie stimmt normalerweise mit der Speicherwortbreite überein.